# Operating Systems Final

## 2017-04-26

- *Four* necessary conditions for a deadlock to arise: *mutual exclusion, hold and wait, no preemption, circular wait*.
- **Resource-allocation graph**:
    - The set of vertices $V$ is partitioned into two different types of nodes: $P$ (all active processes) and $R$ (all resource types).
    - A directed edge $P \rightarrow R$ is called a **request edge**; a directed edge $R \rightarrow P$ is called an **assignment edge**.
    - When a request can be fulfilled, the *request edge* is instantaneously transformed to an *assignment edge*.
    - If the system is in a deadlocked state, then there must exist a cycle in the resource-allocation graph.
    - If each resource type has exactly one instance, then a cycle implies that a deadlock has occurred.
    - Cycles in the graph can be better checked by *BFS* than DFS → $O(n^2)$.
- Methods for handling deadlocks:
    - *Prevent* or *avoid* deadlocks, ensuring that the system will never enter a deadlock state.
    - Allow the system to enter a deadlocked state, *detect* it, and recover.
    - Ignore the problem and pretend that deadlocks never occur in the system.
- Deadlock prevention:
    - Provides a set of methods to ensure that at least one of the four necessary conditions cannot hold.
- Prevention of *mutual exclusion*:
    - At least one resource must be nonsharable to satisfy mutual exclusion.
- Prevention of *hold and wait*:
    - Requires each process to request and be allocated all its resources before it begins execution.
    - Allows a process to request resources only when it has none.
    - Resource utilization may be low, since resources may be allocated but unused for a long period.

  - *Starvation* is possible.
- Prevention of *no preemption*:
  - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
  - Preempt the desired resources from the waiting process and allocate them to the requesting process. While a process is waiting, some of its resources may be preempted, but only if another process requests them.
  - Often applied to resources whose state can be easily saved and restored later.
- Prevention of *circular wait*:
  - Impose a *total ordering* of all resource types and require that each process requests resources in an increasing order of enumeration.
  - An ordering, or hierarchy, does not in itself prevent deadlock.
  - A lock ordering does not guarantee deadlock prevention if locks can be acquired dynamically.
- Deadlock avoidance:
  - Possible side effects of deadlock prevention are low device utilization and reduced system throughput.
  - A deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that a circular-wait condition can never exist.
  - The resource-allocation *state* is defined by the number of available and allocated resources and the maximum demands of the processes.
  - A *safe* state is not a deadlocked state. A deadlocked state is an *unsafe* state.
  - When a process requests an available resource, the system must decide if immediate allocation leaves the system in an unsafe state.
  - Single instance of each resource type → *resource-allocation-graph algorithm*.
  - Multiple instances of a resource type → *banker's algorithm*.
- **Resource-allocation-graph algorithm** → $O(n^2)$
  - **Claim edge**: $P \dashrightarrow R$ indicates that a process may request a resource at some time in the future.
  - Resources must be claimed a *priori* in the system.
  - A claim edge converts to a request edge when a process requests a resource.
  - A request edge converts to an assignment edge when the resource is allocated to the process.
  - When a resource is released by a process, the assignment edge reconverts to a claim edge.

- *Safe state*: no cycle in the in the resource allocation graph.
- The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph.

- **Banker's algorithm** → $O(m \cdot n^2)$ given $n$ processes and $m$ resource types
  - Data structures: `Available[m]`, `Max[n][m]`, `Allocation[n][m]`, `Need[n][m]` (= `Max[n][m]` - `Allocation[n][m]`).
  - *Safe state*: all processes can acquire the maximum needed resources, release the resources, and finish executing regardless of the order.
  - The request can be granted only if (1) Request ≤ Need, (2) Request ≤ Available, and (3) the resulting resource-allocation state is *safe*.
- Deadlock detection:
  - Allows the system to enter a deadlocked state, but provide methods to detect and recover from it.
  - Single instance of each resource type → *wait-for graph*.
  - Multiple instances of a resource type → *banker's algorithm*.
- **Wait-for graph** → $O(n^2)$
  - The resource-allocation graph can be simplified as a *wait-for graph* by removing the resource nodes and collapsing the appropriate edges.
  - To detect deadlocks, the system needs to maintain the wait-for graph and periodically invoke an algorithm that searches for a cycle in the graph.
- Detection algorithm usage:
  - How often to invoke a deadlock detection depends on (1) how often is a deadlock likely to occur? and (2) how many processes will be affected by deadlock when it happens?
  - Every time a request for allocation cannot be granted immediately.
  - Periodically.
  - Whenever CPU utilization drops below a threshold.
  - If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes "caused" the deadlock.
- Recovery from deadlock can be achieved through *process termination* or *resource preemption*.
- Process termination:
  - Two methods: (1) Abort all deadlocked processes. (2) Abort one process at a time until the deadlock cycle is eliminated.
  - If the process was in the midst of updating data, terminating it will leave the data in an

incorrect state.
    - Those processes whose termination will incur the minimum cost should be aborted first.
- Resource preemption:
    - Three issues need to be addressed are (1) selecting a victim, (2) rollback, and (3) starvation.
    - The first issue also occurs in process termination.

## 2017-05-03

- Basic hardware:
    - *Main memory* and the *registers* are the only general-purpose storage that the CPU can access directly.
        - Register is accessible within one CPU clock.
        - Main memory access may take many cycles of the CPU clock. → **Memory stall**.
    - **Cache** is fast memory between the CPU and main memory.
    - Each process has a separate memory space.
    - Legal addresses that a process may access is determined by a **base** and a **limit**.
        - The **base register** holds the smallest legal physical memory address. Also referred to as **relocation register**.
        - The **limit register** specifies the size of the range.
    - CPU hardware compare every address generated in user mode with the registers.
    - Only the operating system can load the base and limit registers by executing privileged instructions in kernel mode.
- Address binding:
    - To be executed, the program must be brought into memory and placed within a process.
    - The processes on the disk that are waiting to be brought into memory for execution form the **input queue**.
    - Addresses in the source program are generally *symbolic*.
    - A *compiler* or *assembler* typically binds these *symbolic addresses* to *relocatable addresses*.
    - The *linkage editor* or *loader* in turn binds the *relocatable addresses* to *absolute addresses*.
    - Each binding is a mapping from one address space to another.
    - The binding of instructions and data can be done at *compile time*, *load time*, or *execution time*.
    - **Compile time**: **absolute code** is generated if where the process will reside in memory at compile time is known a priori.
    - **Load time**: **relocatable code** is generated if where the process will reside in memory is not known at compile time.

    - **Execution time**: binding must be delayed until run time if the process can be moved during its execution from one memory segment to another. Special hardware support is required.
- Logical v.s. physical address space:
    - **Logical address**: An address generated by the CPU. Also referred to as **virtual address**.
    - **Physical address**: An address loaded into the *memory-address register* of the memory and seen by the memory unit.
    - The *compile-time* and *load-time* address-binding methods generate identical logical and physical addresses.
    - The *execution-time* address-binding scheme results in differing logical and physical addresses.
    - The run-time mapping from logical to physical addresses is done by a *hardware* device called the **memory-management unit (MMU)**.
    - The value in the *relocation register* is added to every address generated by a user process at the time the address is sent to memory.
    - A user program deals with logical addresses. It never sees the real physical addresses.
- **Dynamic loading**:
    - A routine is not loaded until it is called.
    - Useful when large amounts of code are needed to handle infrequently occurring cases.
    - Does not require special support from the operating system.
- **Dynamic linking** and shared libraries:
    - **Dynamic linking**: system libraries that are linked to user programs when the programs are run.
    - **Static linking**: system libraries are treated like any other object module and are combined by the loader into the binary program image.
    - **Stub**: a small piece of code that indicates how to locate the appropriate memory-resident library routine or how to load the library if the routine is not already present.
    - The stub replaces itself with the address of the routine, and executes the routine.
    - Generally require help from the operating system.
- Swapping:
    - Swapping makes it possible for the total physical address space of all processes to exceed the real physical memory of the system.
    - Standard swapping involves moving processes between main memory and a **backing store**.

    - The system maintains a **ready queue** consisting of all processes whose memory images are on the backing store or in memory and are ready to run.
    - Lower-priority process is swapped out of the main memory by the dispatcher so higher-priority process can be loaded and executed.
    - The context-switch time in such a swapping system is fairly high.
    - Major part of swap time is *transfer time*. Total transfer time is directly proportional to the amount of memory swapped.
    - Swapping only what is actually used by a process help reduce swap time.
    - If we want to swap a process, we must be sure that it is completely idle.
        - *Solution 1*: never swap a process with pending I/O
        - *Solution 2*: execute I/O operations only into operating-system buffers. → **Double buffering**.
    - Standard swapping is not used in modern operating systems.
    - Modified versions of swapping:
        - *Variation 1*: swapping is normally disabled until the amount of free memory falls below a threshold amount.
        - *Variation 2*: swap only portions of processes rather than entire processes.
    - Mobile systems typically do not support swapping in any form because
        - Mobile devices generally use flash memory rather than more spacious hard disks as their persistent storage.
        - The number of writes that flash memory can tolerate before it becomes unreliable is limited.
        - The poor throughput between main memory and flash memory in these devices.
- **Contiguous memory allocation**:
    - Each process is contained in a single section of memory that is contiguous to the section containing the next process.
    - The memory is usually divided into two partitions: one for the resident operating system and one for the user processes.
    - To place the operating system in either low memory or high memory depends on the location of the interrupt vector.
- Memory protection:
    - **Relocation-register scheme** accomplish memory protection.
    - Allow the operating system's size to change dynamically.
    - **Transient operating-system code** comes and goes as needed.

- Memory allocation:
    - **Partitions**: main memory is divided into several fixed-sized partitions.
    - **Multiple-partition** scheme: Each partition may contain exactly one process. The degree of multiprogramming is bound by the number of partitions.
    - **Variable-partition** scheme: The operating system keeps a table indicating which parts of memory are available and which are occupied.
    - The operating system maintains a list of available block sizes and an input queue.
    - The memory blocks available comprise a set of **holes** of various sizes scattered throughout memory.
    - **Dynamic storage-allocation problem**: How to satisfy a request of size $n$ from a list of free holes? Three strategies are **first-fit**, **best-fit**, and **worst-fit**.
        - **First-fit**: Allocate the first hole that is big enough.
        - **Best-fit**: Allocate the smallest hole that is big enough.
        - **Worst-fit**: Allocate the largest hole.
    - Both *first-fit* and *best-fit* are better than worst fit in terms of decreasing time and storage utilization.
- Memory fragmentation:
    - **External fragmentation**: storage is fragmented into a large number of small holes.
    - **Internal fragmentation**: unused memory that is internal to a partition.
    - The overhead to keep track of this hole will be substantially larger than the hole itself.
    - **50-percent rule**: one-third of memory may be unusable!
    - Two solutions to the problem of external fragmentation are *compaction* and *segmentation/paging*.
    - **Compaction**: shuffle the memory contents so as to place all free memory together in one large block.
    - Compaction is possible only if relocation is dynamic and is done at execution time.
    - Relocation requires only moving the program and data and then changing the base register to reflect the new base address.
    - **Segmentation/Paging**: permit the logical address space of the processes to be noncontiguous.
    - Paging avoids external fragmentation and the need for compaction, whereas segmentation does not.
    - Paging still has some internal fragmentation.
- **Segmentation**:

    - A logical address space is a collection of segments.
    - Logical address consists of a two tuple: **(segment-number, offset)**. Mapped to physical addresses by **segmentation hardware**.
    - **Segment table** is an array of base–limit register pairs indicating the **segment base** and **segment limit** of each segment.
- **Paging**:
    - Breaks physical memory into fixed-sized blocks called **frames** and breaks logical memory into blocks of the same size called **pages**.
    - Logical address is divided into two parts: a **page number** (p) and an **offset** (d). Mapped to physical addresses by **paging hardware**.
    - **Page table** contains the base address of each frame in physical memory.
    - The page size (like the frame size) is defined by the hardware.
    - Small page sizes are desirable. However, overhead is involved in each page-table entry.
    - Clear separation between the programmer's view of memory and the actual physical memory.
    - **Frame table** has one entry for each physical page frame, indicating whether the latter is free or allocated.
    - The hardware implementation of the page table can be kept in registers or main memory.
        - The use of registers for the page table is feasible only for reasonably small size.
        - The use of main memory for the page table requires *two* memory accesses and is therefore slow.
    - **Translation look-aside buffer (TLB)**: a special fast-lookup hardware cache.
    - Some TLBs store address-space identifiers (ASIDs) in each TLB entry.
    - **Address-space identifiers (ASIDs)**:
        - Uniquely identifies each process and is used to provide address-space protection for that process.
        - Allows the TLB to contain entries for several different processes simultaneously.
    - If the page number is not in the TLB (known as a **TLB miss**), a memory reference to the page table must be made.
    - The percentage of times that the page number of interest is found in the TLB is called the **hit ratio**.
    - **Effective memory-access time**: the higher the hit ratio, the less the effective memory-access time.
    - Memory protection in a paged environment is accomplished by *protection bits* associated

with each frame.

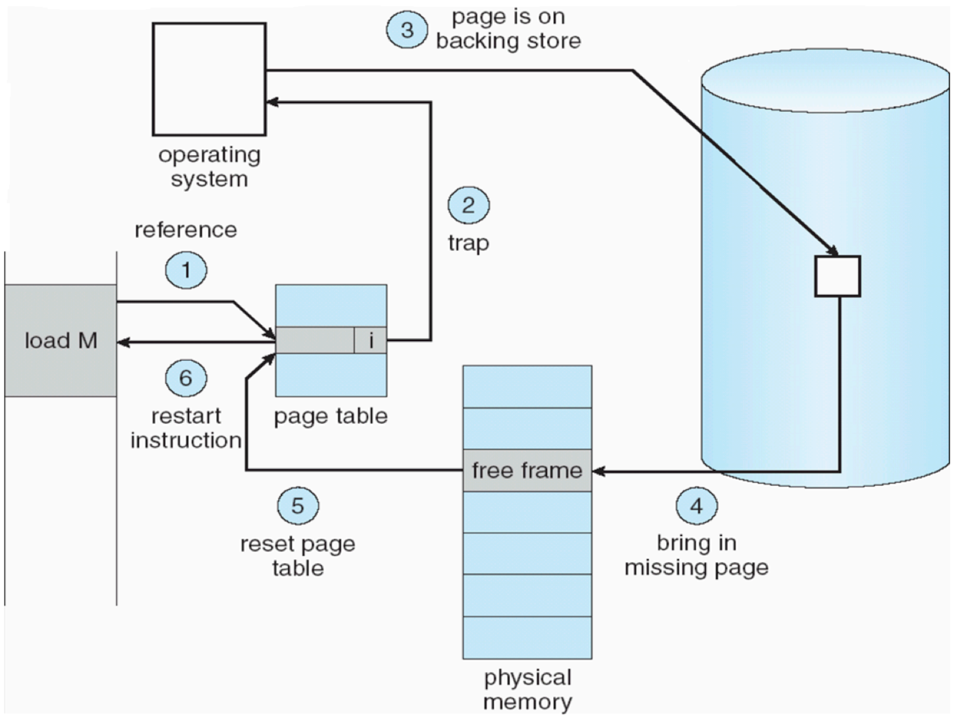
  - **Valid-invalid bit** attached to each entry in the page table indicates whether the associated page is in the process' logical address space.
  - An advantage of paging is the possibility of *sharing* common code, particularly, **reentrant code (pure code)**.
  - **Shared code** must appear in the same location in the logical address space of all processes.
  - Three common techniques for structuring the page table are *hierarchical paging*, *hashed page tables*, and *inverted page tables*.

- **Hierarchical paging**:
  - Breaks up the logical address space into **multi-level page tables**.
  - **Forward-mapped page table**: address translation works from the outer page table inward.
- **Hashed page tables**:
  - Common used in address spaces larger than 32 bits.
  - Hash value is 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.
  - **Clustered page tables**: each entry in the hash table refers to several pages rather than a single page.
- **Inverted page tables**:
  - One entry for each real page (or frame) of memory.
  - Each entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns the page.
  - Inverted page tables often require that an *address-space identifier (ASID)*.
  - Decreases the amount of memory needed to store each page table but increases the amount of time needed to search the table when a page reference occurs.
  - A hash table can be used to limit the search to one or at most a few page-table entries.
  - Difficulty implementing shared memory, which is usually implemented as multiple virtual addresses (one for each process sharing the memory) that are mapped to one physical address.

## 2017-05-10

- **Virtual memory**:
  - The separation of user logical memory from physical memory.

    - **Virtual address space**: the logical (or virtual) view of how a process is stored in memory.
    - Virtual address spaces that include holes are known as **sparse address spaces**.
    - Allows files and memory to be shared by two or more processes through page sharing.
    - Virtual memory can be implemented through *demand paging* or *demand segmentation*.
    - Advantages include less I/O and less memory requirement, faster response, and more user programs.
- **Demand paging**:
    - Loads pages only as they are needed.
    - **Lazy swapper**: never swaps a page into memory unless page will be needed.
    - **Valid–invalid bit scheme**:
        - *Valid*: the page is *both* legal and in memory.
        - *Invalid*: the page either is not valid or is valid but is currently on the disk.
    - While the process executes and accesses pages that are **memory resident**, execution proceeds normally.
    - **Page fault**: Access to a page marked invalid.
    - Steps in handling a page fault:

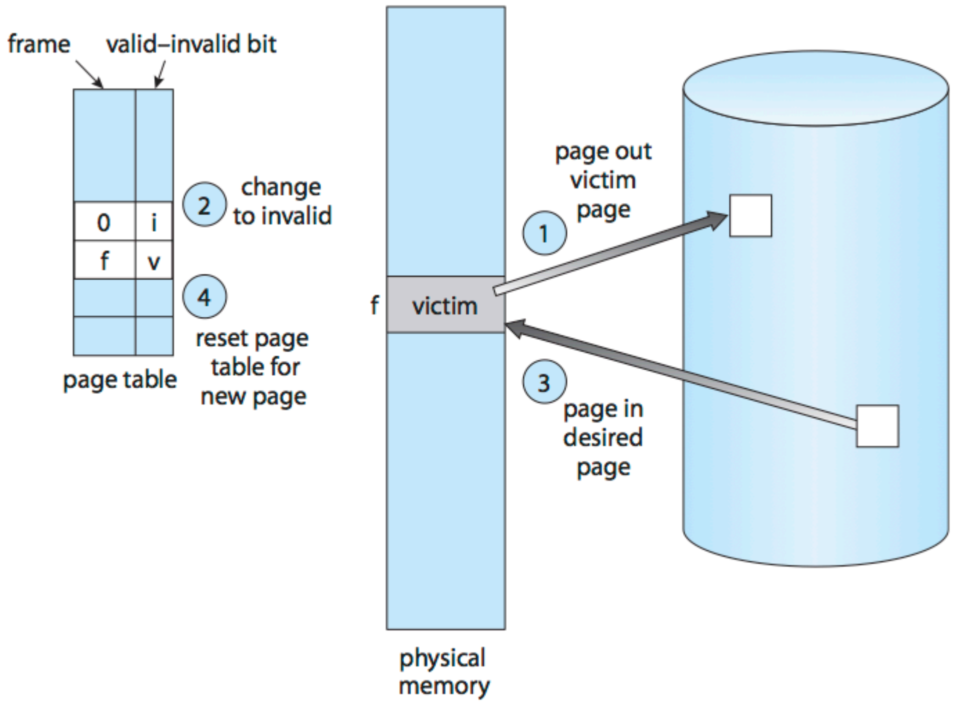


    - **Pure demand paging**: never bring a page into memory until it is required.
    - Programs tend to have **locality of reference**, which results in reasonable performance from demand paging.
    - The hardware to support demand paging is the same as the hardware for paging and swapping: *page table* and *secondary memory* (**swap device** + **swap space**).
    - **Effective access time**: $(1-p) \times$ **memory access** $+ p \times$ **page fault time**, where $p$ is **page**

**fault rate**.

- Three major components of the page-fault service time: (1) service the page-fault interrupt, (2) read in the page, and (3) restart the process.
- A page is brought into the memory from the backing store, either the *swap space* or the file system.
- Disk I/O to swap space is generally faster than that to the file system.
- *Swap space* must still be used for pages not associated with a file, i.e. **anonymous memory**.
- Two major problems to implement demand paging: *frame-allocation algorithm* and *page-replacement algorithm*.

- **Copy-on-write**:
  - Allows both parent and child processes to initially share the same pages in memory after `fork()`.
  - Free pages are allocated from a **pool** of free pages.
  - **Zero-fill-on-demand**: free pages have been zeroed-out before being allocated, thus erasing the previous contents.
  - `vfork()` does not use copy-on-write.
- Page replacement:
  - Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.
  - Steps of page replacement:



  - **Modify bit** (or **dirty bit**): indicates whether a page has been modified or not, and only modified pages are written to disk on page replacement.
  - **Page-replacement algorithm** is to achieve the lowest page-fault rate.

  - **Reference string**: running algorithms on a particular string of memory references and computing the number of page faults.
  - Some algorithms: *FIFO page replacement*, *optimal page replacement*, *LRU page replacement*, *LRU-approximation page replacement*, *enhanced second-chance algorithm*, *counting-based page replacement*, *page-buffering algorithms*, etc.
- **FIFO page replacement**:
  - When a page must be replaced, the oldest page is chosen.
  - **Belady's anomaly**: for some page-replacement algorithms, the page-fault rate may increase as the number of allocated frames increases.
- **Optimal page replacement**:
  - Has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly.
  - Replace the page that will not be used for the longest period of time.
  - Requires future knowledge of the reference string.
- **LRU page replacement**:
  - Replaces the page that has not been used for the longest period of time.
  - May require substantial hardware assistance to reduce the overhead for memory management.
  - Two implementations are feasible: *counters* and *stack*.
    - Counters: Whenever a reference to a page is made, the contents of the clock register are copied to the time-of-use field in the page-table entry for that page.
    - Stack: Whenever a page is referenced, it is removed from the stack and put on the top. Implemented with a *doubly linked list* with a head pointer and a tail pointer.
  - *Does not* suffer from Belady's anomaly.
- **LRU-approximation page replacement**:
  - **Reference bit** is set by the *hardware* whenever that page is referenced but the order of use is never known.
  - Three algorithms: *additional-reference-bits algorithm*, *second-chance algorithm*, and *enhanced second-chance algorithm*.
  - **Additional-reference-bits algorithm**:
    - 8-bit byte for each page in a table is kept in memory.
    - At regular intervals, a timer interrupt transfers control to the operating system.
    - Reference bit for each page is shifted into the high-order bit of its 8-bit byte.
    - Other bits are shifted right by 1 bit and discarding the low-order bit.

        - The page with the lowest number is the LRU page, and can be replaced.
    - **Second-chance algorithm**:
        - FIFO replacement algorithm + reference bit.
        - If the reference bit is 0, we proceed to replace this page.
        - If the reference bit is 1, we give the page a second chance and move on to select the next FIFO page. When a page gets a second chance, its reference bit is cleared, and its arrival time is reset to the current time.
        - Implemented with a *circular queue*.
        - Second-chance replacement degenerates to FIFO replacement if all bits are set.
    - **Enhanced second-chance algorithm**
        - Two tuple: (reference bit, modify bit) can be one of (0,0), (0,1), (1,0), (1,1).
        - (0, 0) is neither recently used nor modified and is the best page to replace.
- **Counting-based page replacement**: least frequently used (LFU) v.s. most frequently used (MFU)
- **Page-buffering algorithms**:
    - The desired page is read into a free frame from the pool before the victim is written out.
    - When the victim is later written out, its frame is added to the free-frame pool.
- Allocation of frames:
    - Keep free frames reserved on the free-frame list at all times.
    - While the page swap is taking place, a replacement can be selected.
    - Each process needs at least a minimum number of pages for better performance.
    - Major allocation schemes: *equal allocation*, *proportional allocation*, and *priority allocation*.
    - **Priority allocation**: the ratio of frames depends not on the relative sizes of processes but rather on the priorities of processes or on a combination of size and priority.
    - *Global/Local replacement*:
        - **Global replacement**: one process can take a frame from another.
        - **Local replacement**: each process select from only its own set of allocated frames.
- **Thrashing**:
    - A process is **thrashing** if it is spending more time paging than executing.
    - When CPU utilization is too low, the OS increase the degree of multiprogramming by introducing a new process to the system.
    - As the degree of multiprogramming increases, CPU utilization also increases, although more slowly, until a maximum is reached.
    - If the degree of multiprogramming is increased even further, thrashing sets in, and CPU utilization drops sharply.

    - Thrashing can be limited (but not solved) by *local replacement algorithm* or *priority replacement algorithm*.
    - Thrashing can be prevented by *locality model* and *working-set model*.
- **Locality model**:
    - A **locality** is a set of pages that are actively used together.
    - As a process executes, it moves from locality to locality.
    - A program is generally composed of several different localities, which may overlap.
- **Working-set model**:
    - Based on the assumption of locality.
    - **Working-set window** $\Delta$ is a fixed number of page references.
    - The set of pages in the most recent $\Delta$ page references is the **working set**.
    - The working set is an approximation of the program's locality.
    - The accuracy of the working set depends on the selection of $\Delta$.
    - Once $\Delta$ has been selected, the operating system monitors the working set of each process and allocates to that working set enough frames to provide it with its working-set size.
    - The number of frames allocated to a process is determined dynamically according to the **page-fault frequency (PFF)**.
    - If the sum of the working-set sizes exceeds the total number of available frames, the OS then still has to select a processes to suspend, replace, and swap out.
- Memory-mapped files:
    - Memory mapping a file is accomplished by mapping a disk block to a page (or pages) in memory.
    - Manipulating files through memory rather than incurring the overhead of using the `read()` and `write()` system calls simplifies and speeds up file access and usage.
    - Multiple processes may be allowed to map the same file concurrently, to allow sharing of data.
- Allocating kernel memory:
    - Kernel memory is often allocated from a free-memory pool.
    - The kernel requests memory for data structures of varying sizes.
    - Pages allocated to user-mode processes do not necessarily have to be in contiguous physical memory.
    - Two strategies: *buddy system* and *slab allocation*.
- **Buddy system**:
    - Allocates memory from a fixed-size segment consisting of physically contiguous pages.

    - Memory is allocated using **power-of-2 allocator**.
    - Request rounded up to next highest power of 2.
    - **Coalescing**: adjacent buddies can be combined to form larger segments quickly.
- **Slab allocation**:
    - **Slab**: one or more physically contiguous pages
    - **Cache**: one or more slabs.
    - There is a single cache for each unique kernel data structure.
    - Each cache is populated with **objects** that are instantiations of the kernel data structure the cache represents.
    - In Linux, a slab may be in one of three possible states: full, empty, or partial.
    - No memory is wasted due to fragmentation.
    - Memory requests can be satisfied quickly.
- **Prepaging**:
    - The large number of page faults that occur when a process is started results from trying to get the initial locality into memory.
    - **Prepaging** is an attempt to prevent this high overhead of initial paging.
    - The strategy is to bring into memory at one time all the pages that will be needed.
    - The OS automatically bring back into memory its entire working set before restarting the process.
- Page size selection must take into consideration fragmentation, table size, I/O overhead, and locality.
- I/O interlock and page locking:
    - When demand paging is used, we sometimes need to allow some of the pages to be locked in memory.
    - If the frame is locked, it cannot be selected for replacement. When the I/O is complete, the pages are unlocked.

## 2017-05-17

- **File**:
    - A named collection of related information that is recorded on secondary storage.
    - Attributes: name, identifier, type, location, size, protection, time, date, user identification.
    - Operations: create, write, read, reposition, delete, truncate.
    - Several pieces of information are associated with an open file: file pointer, open count, disk location of the file, access rights.

        - **File pointer (Current-file-position pointer)**: This pointer is unique to each process operating on the file and therefore must be kept separate from the on-disk file attributes. The same pointer is used by both the read and write operations to save space and reduce system complexity.
        - **Open count**: tracks the number of opens and closes and reaches zero on the last close.
    - *Shared* v.s. *Exclusive* lock:
        - **Shared lock**: a reader lock in that several processes can acquire the lock concurrently.
        - **Exclusive lock**: a writer lock in that only one process at a time can acquire such a lock.
    - *Mandatory* v.s. *Advisory* lock:
        - **Mandatory lock**: the operating system ensures locking integrity.
        - **Advisory lock**: it is up to software developers to ensure that locks are appropriately acquired and released.
    - Access methods: *sequential access, direct access (relative access).*
        - **Sequential access**: information in the file is processed in order, one record after the other.
        - **Direct access (Relative access)**: allows programs to read and write records rapidly in no particular order.
- Disk structure:
    - Partitioning is useful for limiting the sizes of individual file systems.
    - Each partition can hold a separate file system.
    - Any entity containing a file system is generally known as a **volume**.
    - Each volume that contains a file system must also contain information about the files in the system. This information is kept in entries in a **device directory (directory)** or **volume table of contents**.
- **Directory**:
    - A collection of nodes containing information about all files.
    - Operations: search, create, delete, list, rename, traverse.
    - Common logical structures of a directory: *single-level directory*, *two-level directory*, *tree-structured directory*, *acyclic-graph directory*, and *general graph directory*.
    - Each user has his own **user file directory (UFD)**.
    - The **master file directory (MFD)** is indexed by user name or account number, and each

    entry points to the UFD for that user.
  - The sequence of directories searched when a file is named is called the **search path**.
- **Single-level directory**:
  - A single directory for all users may cause naming problem.
- **Two-level directory**:
  - Separate directory for each user solves naming problem among different users.
  - Isolates one user from another but makes sharing files difficult.
  - A user name and a file name define a **path name**.
- **Tree-structured directory**:
  - Allows users to create their own subdirectories and to organize their files accordingly.
  - Absolute v.s. Relative path name.
  - Prohibits the sharing of files or directories.
- **Acyclic-graph directory**:
  - A *shared* directory or file exists in the file system in two (or more) places at once.
  - Allows directories to share subdirectories and files.
  - The UFD of each team member will contain this shared directory of shared files as a subdirectory.
  - Two ways of implementing shared files and subdirectories: *link* and *duplication*
    - **Link**: effectively a pointer to another file or subdirectory and **resolved** by using that path name to locate the real file.
    - **Duplication**: duplicates all information about them in both sharing directories.
  - A file may now have multiple absolute path names.
  - Deletion may leave dangling pointers to the now-nonexistent file.
  - **Symbolic links (Soft links)** are left when a file is deleted, and it is up to the user to realize that the original file is gone or has been replaced.
  - **Nonsymbolic links (hard links)** keeps a reference count in the file information block.
- **General graph directory**:
  - When cycles exist, the reference count may not be 0 even when it is no longer possible to refer to a directory or file.
  - Allows only links to file not subdirectories.
  - Limits arbitrarily the number of directories that will be accessed during a search.
  - **Garbage collection**: traversing the entire file system, marking everything that can be accessed, freeing space that is not marked.
  - Bypasses links during directory traversal.

- File system mounting:
    - A file system must be **mounted** before it can be accessed.
    - **Mount point**: the location within the file structure where the file system is to be attached.
    - On a UNIX system, a file system containing a user's home directories might be mounted as `/home`.
    - Systems impose semantics to clarify functionality.
- Remote file systems:
    - **File transfer protocol (FTP)**: *manually* transferring files between machines via programs.
    - **Distributed file system (DFS)**: remote directories are *automatically* shared and visible from a local machine.
    - World Wide Web: A browser is needed to gain access to the remote files, and separate operations (essentially a wrapper for ftp) are used to transfer files.
    - The machine containing the files is the **server**, and the machine seeking access to the files is the **client**.
    - Once the remote file system is mounted, file operation requests are sent on behalf of the user across the network to the server via the DFS protocol.
    - **Lightweight directory-access protocol (LDAP)**: a secure distributed naming mechanism.
    - Remote file systems can fail due to network failure, server failure, etc.
    - Most DFS protocols either enforce or allow delaying of file-system operations to remote hosts, with the hope that the remote host will become available again.
    - **State information** may be maintained on both the client and the server to recover from failure.
    - **Network file system (NFS)** takes a simple approach, implementing a stateless DFS. A client request for a file read or write would not have occurred unless the file system had been remotely mounted and the file had been previously open.
- Consistency semantics
    - Specify how multiple users are to access a shared file simultaneously.
    - Specify when modifications of data by one user will be observable by other users.
- Protection:
    - **Access-control list (ACL)** specifying user names and the types of access allowed for each user.

## 2017-05-24

- File-system structure:

    - To improve I/O efficiency, I/O transfers between memory and disk are performed in units of **blocks**.
    - Layered file system: application programs → logical file system → file-organization module → basic file system → I/O control → devices.
    - A **file-control block (FCB)** (an **inode** in UNIX file systems) contains metadata information.
- **Virtual file system (VFS)**:
    - An object-oriented techniques to simplify, organize, and modularize the implementation.
    - Separates file-system-generic operations from their implementation by defining a clean VFS interface.
    - Provides a mechanism for uniquely representing a file throughout a network.
- Directory implementation:
    - Linear list: a linear list of file names with pointers to the data blocks.
    - Hash table: The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list.
- Allocation methods:
    - How to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly.
    - Three major methods: *contiguous*, *linked*, and *indexed*.
- **Contiguous allocation**:
    - Each file occupy a set of contiguous blocks on the disk.
    - The number of disk seeks required for accessing contiguously allocated files is minimal.
    - Only the address of the starting block and the length of the area allocated for this file are required.
    - Involves the same *dynamic storage-allocation problem* and *external fragmentation* as in memory allocation.
    - Compacting disks can be done off-line or on-line.
    - Another problem with contiguous allocation is determining how much space is needed for a file.
    - **Extent-based systems**: an **extent** is a chunk of contiguous space.
    - Internal fragmentation can still be a problem if the extents are too large.
    - External fragmentation can become a problem as extents of varying sizes are allocated and deallocated.
- **Linked allocation**:
    - Solves all problems of contiguous allocation.

  - Each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk.
  - It can be used effectively only for sequential-access files, i.e. no random access.
  - Another disadvantage is the space required for the pointers.
  - Collect blocks into multiples, called **clusters**, and to allocate clusters rather than blocks.
  - Adopted by **file-allocation table (FAT)**.
- **Indexed allocation**:
  - Bringing all the pointers together into one location: the **index block**.
  - Supports direct access without suffering from external fragmentation.
  - How large should the index block be?
  - Index block is available with three schemes: *linked scheme*, *multilevel index*, and *combined scheme*.
  - **Combined scheme** adopted by Unix-based systems contains **direct blocks** and **indirect blocks**.
- Free-space management:
  - To keep track of free disk space, the system maintains a **free-space list**.
  - Implementations of free-space list: *bit vector*, *linked list*, *grouping*, *counting*, and *space maps*.
- Efficiency and performance:
  - **Buffer cache**: blocks are kept in cache under the assumption that they will be used again shortly.
  - **Page cache**: uses virtual memory techniques to cache file data as pages rather than as file-system-oriented blocks.
  - **Unified virtual memory**: uses page caching to cache both process pages and file data.
  - **Unified buffer cache**: uses the same page cache to cache both memory-mapped pages and ordinary file system I/O.
  - The memory-mapping call requires using two caches—the page cache and the buffer cache, which is known as **double caching**.
  - **Synchronous writes**: occur in the order in which the disk subsystem receives them, and the writes are not buffered.
  - **Asynchronous writes**: the data are stored in the cache, and control returns to the caller.
  - Sequential access can be optimized by techniques known as *free-behind* and *read-ahead*.
  - **Free-behind**: removes a page from the buffer as soon as the next page is requested.
  - **Read-ahead**: a requested page and several subsequent pages are read and cached.
  - When data are written to a disk file, the pages are buffered in the cache, and the disk

driver sorts its output queue according to disk address.

- Recovery:
    - A system crash can cause inconsistencies among on-disk file-system data structures.
    - At the start of any metadata change, a status bit is set to indicate that the metadata is in flux.
    - If all updates to the metadata complete successfully, the file system can clear that bit.
    - **Consistency checker**: compares the data in the directory structure with the data blocks on disk and tries to fix any inconsistencies it finds.
    - UNIX caches directory entries for reads; but any write that results in space allocation, or other metadata changes, is done synchronously, before the corresponding data blocks are written.
    - **Log structured** (or **journaling**) file systems record each update to the file system as a **transaction**.
    - A **circular buffer** writes to the end of its space and then continues at the beginning, overwriting older values as it goes.
    - A **snapshot** is a view of the file system before the last update took place.
    - Solaris ZFS goes further and provides checksumming of all metadata and data blocks, which assures that data are always correct without any consistency checker.
    - System programs can be used to **back up** data from disk to another storage device and **restore** the data from backup on disk failure.
- **Network file system (NFS)**:
    - Both an implementation and a specification of a software system for accessing remote files across LANs (or WANs).
    - Views a set of interconnected workstations as a set of independent machines with independent file systems.
    - The goal is to allow some degree of sharing among these file systems (on explicit request) in a transparent manner.
    - Designed to operate in a heterogeneous environment of different machines, operating systems, and network architectures.
    - Distinguishes between the services provided by a mount mechanism and the actual remote-file-access services.
    - Accordingly, two separate protocols are specified for these services: a **mount protocol** and a **NFS protocol**.
- Three major layers of NFS architecture:

    - UNIX file-system interface: based on the open, read, write, and close calls:
    - Virtual file system (VFS) layer: distinguishes local files from remote ones, and local files are further distinguished according to their file-system types.
    - NFS service layer: bottom layer of the architecture which implements the NFS protocol.

## 2017-05-31

- Magnetic disks:
    - Components: platter, disk arm, tracks, sectors, cylinder.
    - **Transfer rate**: the rate at which data flow between the drive and the computer.
    - **Positioning time / Random-access time** = **seek time** + **rotational latency**.
    - **Seek time**: the time necessary to move the disk arm to the desired cylinder.
    - **Rotational latency**: the time necessary for the desired sector to rotate to the disk head.
    - **Head crash** results from disk head making contact with the disk surface.
    - A disk drive is attached to a computer by a set of wires called an **I/O bus**, including advanced technology attachment (ATA), serial ATA (SATA), eSATA, universal serial bus (USB), and fibre channel (FC).
    - The data transfers on a bus are carried out by special electronic processors called **controllers**.
    - The **host controller** is the controller at the computer end of the bus.
    - A **disk controller** is built into each disk drive.
    - **Solid-state disks (SSD)**: consume less power and have no moving parts and faster because they have no seek time or latency.
- Disk structure:
    - Modern magnetic disk drives are addressed as large one-dimensional arrays of **logical blocks**.
    - The number of sectors per track is not a constant on some drives.
    - **Constant linear velocity (CLV)**: rotation speed decreases as the head moves from the outer to the inner tracks
    - **Constant angular velocity (CAV)**: the density of bits decreases from inner tracks to outer tracks to keep the data rate constant.
- Disk attachment:
    - **Host-attached storage** is storage accessed through local I/O ports.
    - **Network-attached storage (NAS)** is a special-purpose storage system that is accessed remotely over a data network.

- Disk scheduling:
    - **Disk bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.
    - Algorithms: **FCFS scheduling**, **shortest-seek-time-first (SSTF) scheduling**, **SCAN scheduling**, **circular SCAN (C-SCAN) scheduling**, **LOOK scheduling**.
    - **Shortest-seek-time-first (SSTF) scheduling** selects the request with the minimum seek time from current head position. Essentially a form of shortest-job-first (SJF) scheduling; and like SJF scheduling, it may cause starvation of some requests. Although the SSTF algorithm is a substantial improvement over the FCFS algorithm, it is *not* optimal.
    - **SCAN scheduling / Elevator algorithm**: The disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk.
    - **Circular SCAN (C-SCAN) scheduling**: The head moves from one end of the disk to the other. servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
    - **LOOK scheduling**: Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.
- Selection of a disk-scheduling algorithm:
    - SSTF is common and has a natural appeal because it increases performance over FCFS.
    - SCAN and C-SCAN perform better for systems that place a heavy load on the disk, because they are less likely to cause a starvation problem.
    - Performance depends heavily on the number and types of requests.
    - Requests for disk service can be greatly influenced by the file-allocation method.
    - Either SSTF or LOOK is a reasonable choice for the default algorithm.
    - The Linux **Noop** scheduler uses an FCFS policy but modifies it to merge adjacent requests.
- Disk formatting:
    - **Physical formatting**: Before a disk can store data, it must be divided into sectors that the disk controller can read and write.
    - **Error-correcting code (ECC)**: When the controller writes a sector of data during normal I/O, the ECC is updated with a value calculated from all the bytes in the data area.
    - Before it can use a disk to hold files, the operating system still needs to record its own data structures on the disk.
    - **Partition** the disk into one or more groups of cylinders.
    - **Logical formatting**: creation of a file system.

    - To increase efficiency, most file systems group blocks together into larger chunks, frequently called **clusters**.
    - **Raw disk**: Some operating systems give special programs the ability to use a disk partition as a large sequential array of logical blocks, without any file-system data structures.
    - A tiny bootstrap loader program is stored in the **boot ROM** whose only job is to bring in a full **bootstrap program** from disk.
    - A disk that has a boot partition is called a **boot disk** or **system disk**.
- **Redundant arrays of independent disks (RAID)**:
    - Multiple disk drives provides improvement of reliability via *redundancy*, and improvement in performance via *parallelism*.
    - Mean time to failure, repair, and data loss.
    - **Mean time to repair**: the time it takes (on average) to replace a failed disk and to restore the data on it.
    - **Mirroring**: duplicate every disk (**mirrored volume**) → higher reliability but expensive.
    - **Striping**: **bit-level striping** and **block-level striping** → better performance but not reliable.
    - **Parity**: Each byte in a memory system may have a parity bit associated with it that records whether the number of bits in the byte set to 1 is even (parity = 0) or odd (parity = 1).
- **RAID levels**:
    - Various cost–performance trade-offs and are classified according to levels called **RAID levels**.
    - **RAID level 0**: non-redundant striping → better performance.
    - **RAID level 1**: mirroring → higher reliability.
    - **RAID level 2**: memory-style error-correcting-code (ECC) organization.
    - **RAID level 3**: bit-interleaved parity organization.
    - **RAID level 4**: block-interleaved parity organization.
    - **RAID level 5**: block-interleaved distributed parity.
    - **RAID level 6**: P + Q redundancy scheme.
    - **RAID levels 0+1**: a set of disks are striped, and then the stripe is mirrored to another, equivalent stripe.
    - **RAID levels 1+0**: disks are mirrored in pairs and then the resulting mirrored pairs are striped.

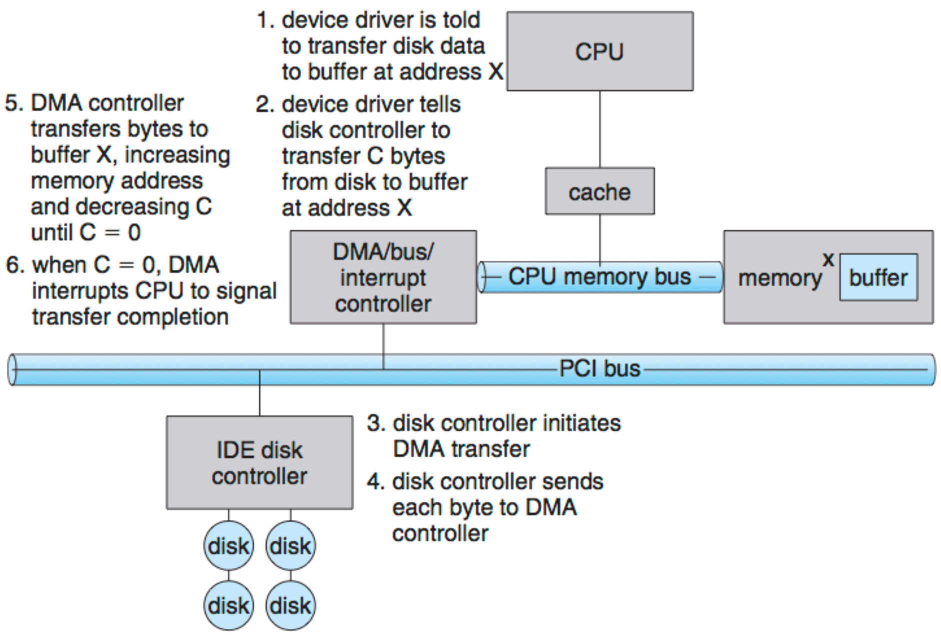
## 2017-06-07

- I/O hardware:
  - **Device driver**: a uniform device-access interface to the I/O subsystem.
  - **Port**: the device communicates with the machine via a connection point.
  - **Bus**: the connection is called a *bus* if devices share a common set of wires.
  - A set of wires and a rigidly defined protocol that specifies a set of messages that can be sent on the wires.
  - **Daisy chain**: I/O devices connected as a chain.
  - **PCI bus**: connects the processor–memory subsystem to fast devices.
  - **Expansion bus**: connects relatively slow devices, such as the keyboard and serial and USB ports.
  - **SCSI bus**: disks are connected together on a *small computer system interface (SCSI) bus* plugged into a SCSI controller.
  - **Controller**: a collection of electronics that can operate a port, a bus, or a device.
  - How can the processor give commands and data to a controller to accomplish an I/O transfer?
  - The controller has one or more registers for data and control signals.
  - The processor communicates with the controller by reading and writing bit patterns in these registers.
  - Special I/O instructions: trigger bus lines to select the proper device and to move bits into or out of a device register.
  - **Memory-mapped I/O**: the device-control registers are mapped into the address space of the processor.
- An I/O port typically consists of four registers:
  - The **data-in register** is read by the host to get input.
  - The **data-out register** is written by the host to send output.
  - The **status register** contains status bits that can be read by the host.
  - The **control register** can be written by the host to start a command or to change the mode of a device.
- Polling:
  - 2 bits are used to coordinate the producer–consumer relationship between the controller and the host.
  - The controller indicates its state through the *busy bit* in the status register.
  - The host signals its wishes via the *command-ready bit* in the command register.
  - 3 CPU-instruction cycles are sufficient to poll a device: read a device register, extract a

  status bit, and branch if not zero.
  - The hardware mechanism that enables a device to notify the CPU is called an **interrupt** to improve efficiency.
- Interrupts:
  - The device controller *raises* an interrupt by asserting a signal on the **interrupt request line**.
  - The CPU *catches* the interrupt and *dispatches* it to the **interrupt handler**.
  - The handler *clears* the interrupt by servicing the device with **interrupt-handler routine** at a fixed address in memory.
  - The basic interrupt mechanism just described enables the CPU to respond to an *asynchronous* event.
  - Three features are provided by the CPU and by the **interrupt-controller hardware**:
    - The ability to defer interrupt handling during critical processing.
    - An efficient way to dispatch to the proper interrupt handler for a device.
    - **Interrupt priority levels**: multilevel interrupts to distinguish between high- and low-priority interrupts.
  - Most CPUs have two interrupt request lines:
    - One is the **nonmaskable** interrupt, which is reserved for events such as unrecoverable memory errors.
    - The second interrupt line is **maskable**: it can be turned off by the CPU.
  - The interrupt mechanism accepts an address, an offset in a table called the **interrupt vector**.
  - **Interrupt chaining**: each element in the interrupt vector points to the head of a list of interrupt handlers.
  - The interrupt mechanism is also used to handle a wide variety of **exceptions**.
  - As for virtual memory paging, a page fault is an exception that raises an interrupt.
  - A user application uses **software interrupt**, or **trap**, to switch from user mode to kernel mode.
  - Interrupts can also be used to manage the flow of control within the kernel.
- **Direct-memory-access (DMA)**:
  - Used to avoid **programmed I/O** for large data movement.
  - To initiate a DMA transfer, the host writes a DMA command block into memory.
  - The CPU writes the address of this command block to the DMA controller, then goes on with other work.
  - The DMA controller proceeds to operate the memory bus directly, placing addresses on the

bus to perform transfers without the help of the main CPU.

- When the entire transfer is finished, the DMA controller interrupts the CPU.
- Steps in a DMA transfer:



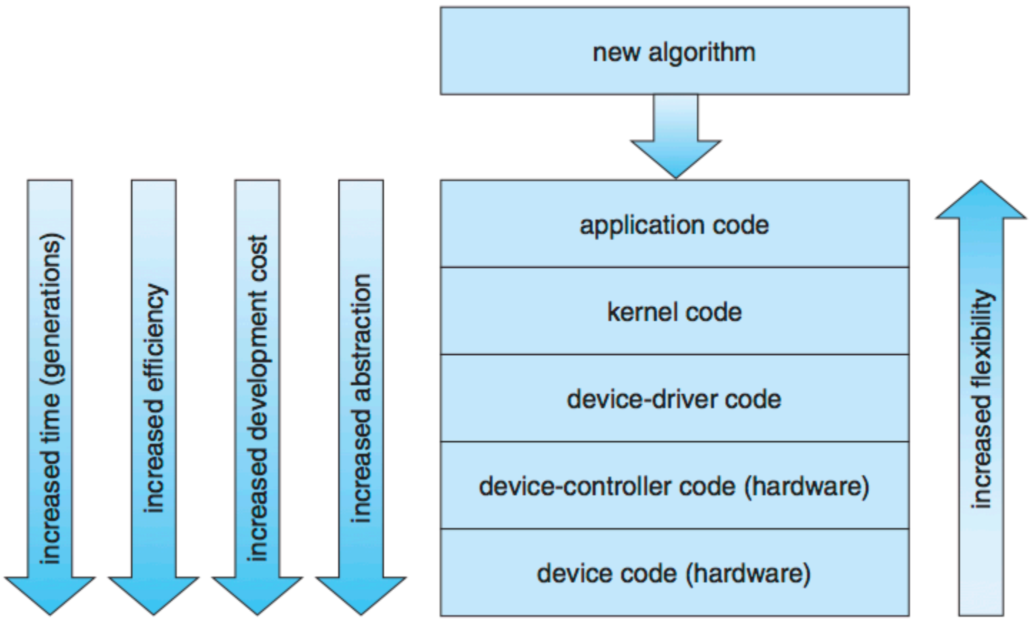
1. device driver is told to transfer disk data to buffer at address X
CPU
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
cache
6. when C = 0, DMA interrupts CPU to signal transfer completion
DMA/bus/interrupt controller
CPU memory bus
memory
x
buffer
PCI bus
IDE disk controller
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
disk
disk
disk
disk

- Application I/O interface:
  - The approach here involves **abstraction, encapsulation, and software layering**.
  - The differences in I/O devices are encapsulated in kernel modules called **device drivers**.
  - The purpose of the device-driver layer is to hide the differences among device controllers from the I/O subsystem of the kernel.
  - Each type of operating system has its own standards for the device-driver interface.
  - Devices vary on many dimensions including
    - Data-transfer mode: character-stream or block.
    - Access method: sequential or random access.
    - Transfer schedule: synchronous or asynchronous.
    - Sharing: sharable or dedicated.
    - Device speed.
    - I/O direction: read–write, read only, or write only.
  - The system call `ioctl()` in UNIX provides an escape (or back door) that transparently passes arbitrary commands from an application to a device driver.
- **Block devices**:
  - Expected to understand commands such as `read()`, `write()`, and also `seek()` if it is a random-access device.
  - **Raw I/O**: to access a block device as a simple linear array of blocks.
  - **Direct I/O**: to allow a mode of operation on a file that disables buffering and locking.

    - A memory-mapped interface provides access to disk storage via an array of bytes in main memory.
- **Character-stream interface**:
    - This interface enable an application to `get()` or `put()` one character.
- Network devices:
    - The network **socket** interface is available in most operating systems.
    - A call to `select()` returns information about which sockets have a packet waiting to be received and which sockets have room to accept a packet to be sent.
    - The use of `select()` eliminates the polling and busy waiting that would otherwise be necessary for network I/O.
- Clocks and timers:
    - Provide current time, elapsed time, timer, etc.
    - **Programmable interval timer**:The hardware to measure elapsed time and to trigger operations.
- Nonblocking and asynchronous I/O:
    - **Blocking**: process suspended until a blocking I/O completed.
    - **Nonblocking**: a nonblocking I/O returns immediately with whatever data are available.
    - **Asynchronous**: an asynchronous I/O returns immediately but requests a transfer that will be performed in its entirety.
- **Vectored I/O**:
    - **Scatter–gather** method.
    - Allows one system call to perform multiple I/Ooperations involving multiple locations.
    - The system call accepts a vector of multiple buffers.
- I/O scheduling:
    - Implement scheduling by maintaining a wait queue of requests for each device.
    - When a kernel supports asynchronous I/O, it must be able to keep track of many I/O requests at the same time.
    - The operating system might attach the wait queue to a **device-status table**.
- Buffering:
    - To cope with a speed mismatch between the producer and consumer of a data stream.
    - **Double buffering**: decouples the producer of data from the consumer.
    - To provide adaptations for devices that have different data-transfer sizes.
    - To support **copy semantics** for application I/O.
    - Copying of data between kernel buffers and application data space provides clean

semantics.

- Caching:
    - **Cache**: a region of fast memory that holds copies of data.
- **Spooling** and device reservation:
    - **Spool**: a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams.
    - The spooling system copies the queued spool files to the device one at a time.
    - Device reservation provides exclusive access to a device.
- Device functionality progression:



## 2017-06-14

- **Principle of least privilege**: programs, users, and even systems be given just enough privileges to perform their tasks.
- **Need-to-know principle**: a process should be able to access only those resources that it currently requires to complete its task.
- **Protection domain**: specifies the resources that the process may access.
- **Access right**: the ability to execute an operation on an object.
- The association between a process and a domain may be either *static*, if the set of resources available to the process is fixed throughout the process's lifetime, or *dynamic*.
- **Domain switching** enables the process to switch from one domain to another.
- Each user, process, or procedure may be a domain.
- In UNIX, an owner identification and a domain bit (known as the **setuid bit**) are associated with each file.

- When the setuid bit is on, and a user executes that file, the userID is set to that of the owner of the file.
- The general model of protection can be viewed abstractly as a matrix, called an **access matrix**.
- The rows of the access matrix represent *domains*, and the columns represent *objects*.
- **Confinement problem**: the problem of guaranteeing that no information initially held in an object can migrate outside of its execution environment.
- Access matrix can be implemented as **access lists** for objects or **capability lists** for domains.
- Forms of security violations: breach of confidentiality, breach of integrity, breach of availability, theft of service, denial of service, etc.
- Methods of security violattions: masquerading, replay attack, man-in-the-middle attack, session hijacking, etc.
- Authorized users may be tricked into allowing access via **social engineering**, including phishing, dumpster diving, etc.
- Security must occur at four levels to be effective: physical, human, operating system, and network.
- Program threats as a **back-door daemon** include Trojan horse, trap door, logic bomb, stack and buffer overflow, viruses, etc.
- **Trojan horse**: a code segment that misuses its environment.
- Stack and buffer overflow:
  - Overflow an input field, command-line argument, or input buffer until it writes into the stack.
  - Overwrite the current return address on the stack with the address of the exploit code loaded in step 3.
  - Write a simple set of code for the next space in the stack that includes the commands that the attacker wishes to execute.
- Viruses: malicious code fragment embedded in legitimate program.
- **Virus dropper** inserts virus onto the system.