Interrupt. Interrupts allow a device to change the flow of control in the CPU. Hardware may trigger an interrupt at any time by sending a signal to CPU. Software may trigger an interrupt either by an error or by a user request for an operating system service (trap). Interrupt transfers control to the interrupt service through interrupt vector, which contains the addresses (function pointer) of all the service routines. Architecture must save the address of the interrupted instruction. Incoming interrupts are disabled.

Dual-mode operation. Provide hardware support to differentiate between user mode (1), kernel mode (0). Privileged instructions, executed only in monitor mode, requested by users (system calls). All I/O instructions are privileged instructions. Memory protection: Protect interrupt vector and the interrupt service routines, data access and overwrite from other programs. HW support, base reg & limit reg. CPU protection: Prevent user program from not returning control. HW support, timer, interrupt after a specified period. Protection: CPU has very limited capability when executing in user mode.

System structure. Layered, access services from inner layers, easy to implement, hard to define layers. Microkernel, leave to user space, message passing, easy to extend. Modular, object-oriented, separate components, call any service, communication.

Process. A program in execution in memory. It includes: text section, data section, stack (temporary local vars and fcns), heap (dynamic allocated vars or classes), current activity (program counter, register contents), a set of associated resources (e.g. open file handlers). Threads. All threads belonging to the same process share text section, data section, and OS resources. But each thread has its own thread ID, program counter, register set, and a stack. Process Control Block (PCB): Process state, Program counter, CPU registers, CPU scheduling information (e.g. priority), Memory-management information (e.g. base/limit register), I/O status information, Accounting information. Process states: New, Ready, Running, Waiting, Terminated. Context Switch: Kernel saves the state of the old process and loads the saved state for the new process. Context-switch time is purely overhead. Switch time depends on memory speed, number of registers, existence of special instructions, hardware support.

Schedulers. Job queue (New). Ready queue (Ready). Device queue (Wait). Short-term scheduler: Ready, Run. Long-term: New, Ready; Degree of multiprogramming, Good mix of CPU-bound & I/O-bound processes. Medium-term: Ready, Wait; Swap out: removing processes from memory, Swap in. *Process creation.* resources sharing (all, subset, none); execution (concurrently, wait until termination), address space (duplicate, loaded program).

Interprocess Communication. information sharing, computation speedup, convenience, modularity. Methods: shared memory (memory addr access, synchronization, deadlock, cache coherence), message passing (direct: name explicitly; indirect: mailbox; blocking: wait (zero capacity buffer)), sockets (IP & port num, unstructured stream of bytes, data parsing), remote procedure call (networked systems, stubs (client-side proxy), packs parameters into a message (parameter marshaling)), pipes (specific type of file; ordinary, parent process writes child reads, unidirectional; named, several processes).

Multithread. Benefits: Responsiveness (continue running even if blocked), Resource sharing (same memory addr), Utilization of MP arch (parallel), Economy (faster context switch). Challenges: divide & balance, data dependency. User/Kernel threads, library/kernel(OS) mgt. User thread blocks, entire process blocks; while kernel schedules another one.Multithreading Models. Many-to-one, many user-level threads to one kernel thread, usually on systems with no kernel threads, block issue, no parallel. One-to-one, limited num, more concurrency. Many-to-many, more user threads, parallel, no block. Thread cancellation, asynchronous cancel immediately; defer cancellation, safe. Signal handling, synchronous to the same process. Thread pool, faster to service a request than creation, allows number of applications to be bound to the size of the pool. Thread specific data, own copy of data, useful when no control over creation. Upcalls, from kernel to thread library (num of threads getter).

Scheduling. Decisions occur: running wait, running ready, wait ready, terminate. Preemptive, all cases (Mac OS X, win95 after); Non-Preemptive, 1&4. Preemptive issues, inconsistent state of shared data, process in the middle of critical changes. Solutions: process synchronization, wait for the completion. Dispatch latency: schedule, interrupt re-enable, context switch.

Scheduling criteria. CPU utilization, Throughput (number of completed processes per time unit), Turnaround time (submission to completion), Waiting time (waiting time in the ready queue), Response time (submission to the first response).

Scheduling algs. First-Come-First-Served, Convoy effect: short processes behind a long process. Shortest-Job-First, burst length, Wait time = completion time – arrival time – run time, approximate length by exponential avg. Priority Scheduling, process specific, aging (increase priority as time goes). Round-Robin, time quantum, large (FIFO), small (context switch overhead increases), higher avg turnaround, better response. Multilevel Queue, different queue with different schedules, additional schedule between queues. Multilevel Feedback Queue, process move between queues, I/O bound high priority, aging. E.g. Windows XP, Multilevel feedback queue, Round-robin in each priority queue. Linux, Preemptive priority based, allows only user mode processes to preempt.

Multiprocessor scheduling. Asymmetric multiprocessing, master slave. Symmetric multiprocessing, self schedule, need synchronization mechanism. Processor affinity. A process has an affinity for the processor on which it is currently running, cache invalidation and repopulation has high cost, nonuniform memory access (NUMA). Load balancing, push migration (high to low), pull, depends on overall load, tradeoff with processor affinity. Multi-Core processor, Intel hyper-threading utilizes memory stall. Coarse-grained, flush when memory stall; fine-grained, additional hardware for thread switch. *Real-time scheduling.* Keeping deadlines. Algs for soft. Rate-Monotonic (RM), shorter period, higher priority, fixed priority. Earliest-Deadline-First (EDF).

Address binding. Compile time, absolute code. Load time, relocatable code: can be run from any memory location. Execution time, virtual address code, most OS, MMU. *Load, link.* Dynamic loading, a routine is loaded into memory when it is called, better memory-space utilization, no special support from OS is required. Static linking: libraries are combined by the loader into the program in-memory image, duplicated code. Dynamic linking: linking postponed until execution time, only one code copy in memory and shared by everyone, a stub is included in the program in-memory image for each lib reference.

Swapping. Swap out processes to the backing store on disk (separated from file sys), midterm scheduling, free up mem, high priority schedule. Compile/load time address binding, swap back to the same address; execution time, no need. A process to be swapped must be idle, consider pending I/O, consider preemptive.

Contiguous memory allocation. Fixed-partition allocation, internal fragmentation (internal space not used). Variable-size partition, first-fit, best-fit (smallest hole), worst-fit (largest), external fragmentation (total is big, not contiguous). Solution to fragmentation: compaction, shuffle at execution time, for execution time address binding only.

Non-contiguous, Paging. No external frag, limited internal frag, shared pages. Frames: fixed-sized blocks in physical memory. Pages: fixed-sized (4,8KB) blocks in logical address space. Page table, translate logical to physical addresses, maintained by OS for each process, stored in memory, addr kept by Page-table base register (PTBR), change when context switch. HW support: Translation Look-aside Buffers (TLB), associative memory, in cache. Flushed after context switch, or TLB entry must has a PID field (address-space identifiers (ASIDs)). Frame table, by OS, indicate whether a frame is free or allocated. Page table memory structure. Hierarchical, more mem access. Hashed, chained. Inverted page table, frame table, (pid, page num). Shared pages, reentrant code

(const), one copy of the shared code needs to be kept in physical memory. Mem protection. Protection bit. Valid bit. PTLR to avoid waste. *Non-contiguous, Segmentation.* Logical addr, (seg num, offset). Segmentation table, (base, limit). STBR, physical addr of table. STLR, num of segs. Protection bits with segs. Sharing by having the same base. MMU: Segmentation in logical address space, paging in physical address space.

Demand paging. A page is brought into mem when needed. Reference to a page: invalid, abort; not in mem, pager paging. Pure demand paging starts a process with no page. HW support: Page table, valid bit. Secondary mem (swap space, backing store). Page fault. First reference to a page. Handling steps: reference, trap, page is on backing store, page replacement, restart instruction. Page replacement. Page fault and no free frame.

Process creation. Copy-on-Write: parent and child process share the same frames initially, copy frame when either modifies. Free frames are from a page pool, fill zero first when demanded. Memory-Mapped Files. A file is initially read using demand paging. Subsequent reads/writes to/from the file are treated as ordinary memory accesses. Faster file access, shared files.

Page replacement. Steps: free frame exists; swap out victim frame, invalid; swap in desired page, valid. First-In-First-Out, Belady's Anomaly: more frames, more faults. Optimal(Belady), replace the page that will not be used for the longest period of time, needs future knowledge. Least Recently Used (LRU), page that has not been used for the longest period of time, additional HW support. Stack algorithm: the set of pages in memory for n frames is always a subset of the set of pages that would be in memory with n+1 frames, optimal, LRU. Counting algorithms, least/most frequently used, not common.

Frame allocation. Fixed allocation, equal allocation, proportional allocation (process size). Priority allocation, proportional allocation based on priority. Local allocation: each process select from its own set of allocated frames. Global allocation: all frames in mem, better performance.

Thrashing. A process is thrashing if it is spending more time paging than executing (not enough frames for active pages). Queued for I/O, low CPU utilization, OS increases the degree of multiprogramming, loop. Process locality: a set of pages that are actively used together (program structure (subroutine, loop, stack), data structure (array)). Locality model: as a process executes, it moves from locality to locality. Working-set model, working-set window w, working-set: set of pages in most recent w page references, total working-set size for all processes, expensive. Page fault frequency scheme, allocate frame if exceeds max freq, remove if falls min. Figure. New locality, peak; working-set in mem, decrease.

File. A logical storage unit created by OS. Identifier, Name, Type, Location, Size, Protection, Last-access time, Last-updated time. Open-file attributes (metadata): File pointer (per-process table), File open count (system-wide table), Disk location (sys table), Access rights (pp). Pp table points to sys table. Access method. Sequential access, read/write next (block), reset, skip/rewind. Direct(relative) access, arbitrary position. Index access, contains pointers to blocks of a file.

Directory structure. Raw. Formatted partition with file system is called volume. Directories are used by file system to store the information about the files in the partition. Directory structure and the files reside on disk. Single-Level Directory, filename has to be unique, poor efficiency in locating. Two-level, separate dir for each user. Tree-Structured, Absolute path: starting from the root; Relative path: starting from a directory. Allow user define directories. Acyclic-Graph, use links to share files or directories (pro), multiple absolute paths, delete when reference counter is 0 (con). General-Graph, garbage collection reallocates.

File system mounting, sharing. A file system must be mounted before it can be accessed. Mount point: the root path that a FS will be mounted to. Mount timing: boot time, automatically at run-time, manually at run-time. Acess control list. Each user: (userID, groupID). Each file has 3 sets of attributes: owner, group, others. group/others attributes are set by owner or root. ACL for each user, bad if users are unlimited. 3 classes of users, 3 sets of ACL (RWX) for each file.

FS structure. I/O transfers between memory and disk are performed in units of blocks, one block is one or more sectors, one sector is usually 512 bytes. Two design problems: interface to user programs, interface to physical storage (disk).

FS implementation. On-Disk Structure: Boot control block (per partition), info needed to boot an OS, typically the first block; boot block. Partition control block (per partition), num of blocks, block size, free-block-list, free FCB pointers; superblock. File control block (per file), permissions, size, location of data blocks; inode. Directory structure (per file system). In-Memory Structure: Partition table, each mounted partition info. Directory structure, recently accessed directories. System-wide open-file table: copy of each opened file's FCB. Per-process open-file table: pointer (file handler/descriptor) to the corresponding entry in the above table. File creation: App calls logical FS, logical FS allocates a new FCB, reads in the corresponding directory structure into memory, updates the dir structure with the new file name and the FCB. Open, read, write, search sys table with filename, if not existed, search directory, copy FCB to sys table, keep track of open count, pp table points to FCB, return pointer. Utilize this pointer to read/write. Virtual FS, provides an object-oriented way, allows the same system call interface to be used for different types of FS.

Allocation methods. How disk blocks are allocated for files. Contiguous Allocation, disk seeks are minimized, sequential & random access. external fragmentation (compaction), file cannot grow. Extent-Based, an extent is a contiguous blocks of disks. Linked, each block contains a pointer to the next block, sequential-access (cluster of blocks), reliability. Indexed, directory tracks the addr of the file index block, index block stores block num for files. *Free space mgt.* Free-space list: records all free disk blocks. Bit Vector (bitmap), HW support, bit operations. Linked List, keep the first free block pointer in a special location on the disk and caching in memory. Grouping, pointers to free blocks and a pointer to next grouping. Counting, addr, num of free blocks.

I/O overview. Port: A connection point between I/O devices and the host. Bus: A set of wires and a well-defined protocol that specifies messages sent over the wires. Controller: A collection of electronics that can operate a port, a bus, or a device. Port-mapped I/O, Use different address space from memory. Memory-mapped I/O reserve specific memory space for device. Poll (busy-waiting): processor periodically check status register of a device. Interrupt: device notify processor of its completion. Programmed I/O: transfer controlled by CPU. Direct memory access (DMA) I/O: controlled by DMA controller, commonly used for memory-mapped and interrupt I/O. Process: device driver is told to transfer disk data to buffer at addr X, device driver tells disk controller to transfer C bytes from disk to buffer at addr X, disk controller initiates DMA transfer, disk controller sends each byte to DMA controller, DMA controller transfers bytes to buffer X, DMA interrupts CPU to signal completion.

Kernel I/O subsystem. I/O subsystem: the methods to control all I/O devices. I/O Scheduling, order the jobs in I/O queue. Buffering, store data in memory while transferring between I/O devices. Caching, fast memory that holds copies of data. Spooling, holds output for a device. Error handling. I/O protection, privileged instructions. Improve performance, reduce context switch; reduce data copying; reduce interrupts by using large transfers, smart controllers, polling; DMA; balance for highest throughput.

Application I/O interface. Device drivers: a uniform device-access interface to the I/O subsystem, hide the differences among device controllers. Back-door interfaces enable an application to access any functionality implemented by a device driver without the need to invent a new system call. Interrupt handshake. Device asserts interrupt request (IRQ), CPU checks IRQ line, save status, search interrupt vector table for routines, fetch next instruction from the routine, restore interrupted process after execution.