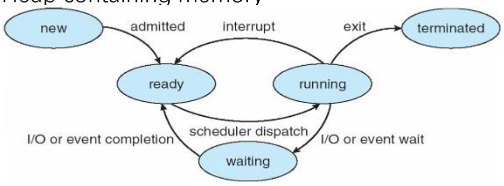
Interrupt: Inform of an event of hardware or software. CPU stop the current thing and execute the interrupt program, then back to pre.

What is interrupt? The occurrence of an event is usually signaled by an interrupt from either the hardware or the software. Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by way of the system bus. Software may trigger an interrupt by executing a special operation called a system call (also called a monitor call). An interrupt is a signal emitted by a device attached to a computer or from a program within the computer. It requires the operating system (OS) to stop and figure out what to do next. An interrupt temporarily stops or terminates a service or a current process. Most I/O devices have a bus control line called Interrupt Service Routine (ISR) for this purpose. 1. Interrupt transfers control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all the service routines. 2. Interrupt architecture must save the address of the interrupted instruction 3. A trap or exception is a software-generated interrupt caused either by an error or a user request 4. An operating system is interrupt driven.

DMA: Used for high-speed I/O devices able to transmit information at close to memory speeds

multiprogramming
Single programming cannot keep CPU and I/O devices busy at all times. Multiprogramming organizes jobs (code and data) so CPU always has one to execute. A subset of total jobs in system is kept in memory, a job selected and run via **job scheduling**. When it has to wait (for I/O for example), OS switches to another job.
Timesharing (multitasking)?
CPU switches jobs so frequently that users can interact with each job while it is running, creating interactive computing with response time < 1s. Do CPU scheduling, if several jobs ready to run at the same time and processes don't fit in memory.
process management:
Creating and deleting both user and system processes Suspending and resuming processes Providing mechanisms for process synchronization Providing mechanisms for process communication Providing mechanisms for deadlock handling
memory management
- Keep track of which part of memory are currently being used by whom
- Decide what part of process data move in and move out
- Allocating and deallocating memory space as needed.
storage management:
- Free-space management.
- Storage allocation.
- Disk scheduling.
CLI or **command interpreter** allows direct command entry
System calls: Programming interface to the services provided by the OS
Why use APIs rather than system calls?
- good portablilty, the program can compile and execute on any system with the same API
- For programer, system call is more difficult than API, because it focus on details
System Programs: Provide a convenient environment for program development and execution. Some of them are simply user interfaces to system calls; others are considerably more complex
Mechanisms determine how to do, **policies** decide what will be done. The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later
Layered Approach:
divided a number of layers (levels), each built on top of lower layers. bottom layer hardware; the highest the user interface. layers are uses functions of only lower-level layers

Microkernel System: Moves as much from the kernel into user space
Communication takes place between user modules using **message passing**
Benefits:
Easier to extend a microkernel, to port the operating system to new architectures
More reliable (less code is running in kernel mode. More secure
Detriments: Performance overhead of user space to kernel space communication
loadable kernel modules: Uses object-oriented approach. core component is separate. talks to the others over known interfaces. loadable as needed within the kernel
Process: a program in execution; process execution must progress in sequential fashion. The program code Current activity including program counter, processor registers Stack data Heap containing memory



Process Control Block (PCB): Information of Process state, Program counter, CPU registers CPU scheduling information, memory allocated to the process. Accounting information – CPU used, clock time elapsed since start, time limits I/O status information – I/O devices allocated to process, list of open files
Process scheduler selects among available processes for next execution on CPU
Ready queue – set of all processes residing in main memory, ready and waiting to execute
Device queues processes waiting for an I/O device
Long-term scheduler – selects which processes should be brought into the ready queue
Short-term scheduler– selects which process should be executed next and allocates CPU
Long controls the degree of **multiprogramming**
If the multiprogramming degree is stable, the speed of creating a process is similar to the speed of process leaving the system. Therefore, only when the process leave the system, it needs the long-term scheduler. Because the time between executing 2 process is rather long, the long-term scheduler can afford more time to decide which process to add into the ready queue.
Medium-term scheduler if degree of multiple programming needs to decrease. Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**
Context Switch: When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch
If no parent waiting, then terminated process is a **zombie**
If parent terminated, processes are **orphans**
shared memory build a area of cooperating memory space to read and write in it. **massage passing** exchange message in cooperating area.
message passing is useful in exchanging **small data**, because no need to avoid conflict, and it is easier to implement.
share memory is faster, because it doesn't use the system call when data exchange. It only used the system call when it build the share memory space.
Thread: a basic unit of CPU utilization, contains thread ID, program counter, a group of register, and stack. It shares code, data and other operating system resource with other threads in the same process.
Process creation is heavy-weight while **thread** creation is light-weight
Thread benefits: **Responsiveness –** May allow continued execution if part of process is blocked, especially important for user interfaces
Resource Sharing – Threads share resources of process, easier than shared memory or message passing
Economy – Cheaper than process creation, thread switching lower overhead than context switching
Scalability – Process can take advantage of multiprocessor architectures
Parallelism implies a system can perform more than one task simultaneously ***Concurrency*** supports more than one task making progress
Amdahl's Law
Many to one: one block all block

$$speedup \le \frac{1}{S + \frac{(1-S)}{N}}$$

Thread Pool: Create a number of threads in a pool where they await work
faster to service a request with an existing thread than create a new thread
allow bound to the size of the pool creating task allows different strategies for running task
base and **limit registers** define the logical address space
Binding
Compile time: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes
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
Memory-Management Unit: Hardware device that at run time maps virtual to physical address
Dynamic **relocation** using a relocation register
One program is only loaded when it is being called. All the program are saved on disk in a reloadable manner. When a program is called, it will check if it is on memory, and if not, will load. The program can be large, but the load part is small.
Stub: Small piece of code used to locate the appropriate memory-resident library routine
First-fit: the **first** hole that is big enough
Best-fit: the ***smallest*** hole that is big enough; must search entire list, unless ordered by size
Worst-fit: Allocate the ***largest*** hole;
External Fragmentation – total memory space exists to satisfy a request, but it is not contiguous
Internal Fragmentation – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
Compaction: Shuffle memory contents to place all free memory together in one large block
Segmentation； Memory-management scheme that supports user view of memory
Paging: Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
Set up a **page table** to translate logical to physical addresses
TLB: a special fast-lookup hardware cache
Inverted Page Table Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages
Virtual memory – separation of user logical memory from physical memory. Only part of the program needs to be in memory for execution. Logical address space can therefore be much larger than physical address space. Allows address spaces to be shared by several processes. Allows for more efficient process creation. More programs running concurrently. Less I/O needed to load or swap processes
Lazy swapper – never swaps a page into memory unless page will be needed
Demand Paging - page fault

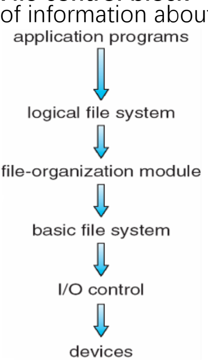
1. Trap to the operating system
2. Save the user registers and process state
3. Determine that the interrupt was a page fault
4. Check that the page reference was legal and determine the location of the page on the disk
5. Issue a read from the disk to a free frame:
    1. Wait in a queue for this device until the read request is serviced

2. Wait for the device seek and/or latency time
3. Begin the transfer of the page to a free frame

6. While waiting, allocate the CPU to some other user
7. Receive an interrupt from the disk I/O subsystem (I/O completed)
8. Save the registers and process state for the other user
9. Determine that the interrupt was from the disk
10. Correct the page table and other tables to show page is now in memory
11. Wait for the CPU to be allocated to this process again
12. Restore the user registers, process state, and new page table, and then resume the interrupted instruction

Copy-on-Write (COW) allows parent and child processes to initially *share* the same pages
Frame-allocation algorithm how many frames to give each process and which frames to replace
Page-replacement algorithm want lowest page-fault rate on both first access and re-access
Page-Buffering Algorithms give out free frame in pool before find victim frame.
Thrashing ≡ a process is busy swapping pages in and out. Low CPU utilization. Operating system o increase the degree of multiprogramming. Another process added to the system
size of locality > total memory size
Page-Fault Frequency If actual rate too low, process loses frame If actual rate too high, process gains frame
Memory-Mapped Files: Memory-mapped file I/O allows file I/O to be treated as routine memory access by mapping a disk block to a page in memory. A page-sized portion of the file is read from the file system into a physical page. Subsequent reads/writes to/from the file are treated as ordinary memory accesses
Buddy System Allocates memory from fixed-size segment contiguous page using power-**2**
page size trade off
small page size: Fragmentation, Locality, Resolution
big page size: Page table size I/O overhead, Number of page faults, TLB size & effective

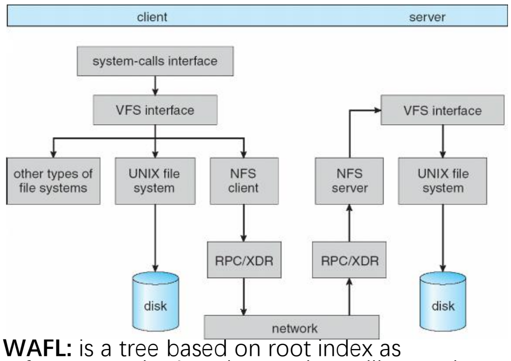
File system: resides on secondary storage (disks)
1Provided user interface to storage, mapping logical to physical
2 Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
File control block – storage structure consisting of information about a file



Device drivers manage I/O devices at the I/O control layer
Basic file system manages memory buffers and caches
File organization module Translates logical block to physical block
Manages free space, disk allocation
Logical file system manages metadata information

File-System Implementation
Boot control block boot OS from that volume
Volume control block (superblock, master file table) Total # of blocks, free blocks, block size
File Control Block (FCB) contains details about the file Inode number, permissions, size, dates
Virtual File Systems
allows the same system call interface (the API) to be used for different types of file systems
The API is to the VFS interface, rather than any specific type of file system
Contiguous allocation
external fragmentation, need for compaction
Linked allocation
Improve efficiency by clustering blocks into groups but increases internal fragmentation

Locating a block can take many I/Os and disk seeks
Indexed Allocation
no external fragmentation, but have overhead of index block
Combined Scheme
Free-Space Management - bit map
page cache caches pages rather than disk blocks using virtual memory techniques and addresses
Buffer cache – separate section of main memory for frequently used file blocks
Synchronous writes No buffering / caching – writes must hit disk before acknowledgement
Asynchronous writes: data store in cache, and return the control to caller.
A **page cache** caches pages rather than disk blocks using virtual memory techniques and addresses
unified buffer cache uses the same page cache to cache both memory-mapped pages and ordinary file system I/O to avoid **double aching**
Log structured (or **journaling**) file systems record each metadata update to the file system as a **transaction** written to a log
If the file system crashes, all remaining transactions in the log must still be performed
NFS Interconnected workstations viewed as a set of independent machines with independent file systems, which allows sharing among these file systems in a transparent manner
Virtual File System (VFS) layer distinguishes local files from remote ones, and local files are further distinguished according to their file-system types
The VFS activates file-system-specific operations to handle local requests according to their filesystem types Calls the NFS protocol procedures for remote requests
NFS service layer bottom layer of the architecture



WAFL: is a tree based on root index as reference node. Any data update will move in new block instead of replacing the current block. New inode pointes to new data, old inode (snapshot) point to old block

Access Latency = **Average access time** = average seek time + average latency
SSD: No moving parts, seek, rotational latency
SCSI (Small Computer System Interface) itself is a bus, up to 16 devices on one cable, **SCSI initiator** requests operation and **SCSI targets** perform tasks Each target can have up to 8 logical units
Network-attached storage (**NAS**) is storage made available over a network rather than over a local connection (such as a bus)
Remotely attaching to file systems
iSCSI (Internat SCSI) protocol uses IP network to carry the SCSI protocol
Remotely attaching to devices (blocks)
Storage Area Network
Multiple hosts attached to multiple storage arrays – flexible

Disk Scheduling
FCFS: First come first serve.
SSTF: Shortest Seek Time First selects the request with the minimum seek time from the current head position
SCAN: The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues. But if requests are uniformly dense, largest density at other end of disk and those wait the longest
C-SCAN: 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
C-Look: without first going all the way to the end of the disk
SCAN and C-SCAN better for systems that place a heavy load on the disk Less starvation

Disk Management
Low-level formatting, or **physical formatting** — Dividing a disk into sectors that the disk controller can read and write
still needs to record its data structures on the disk **Partition** the disk into one or more groups of cylinders, each treated as a logical disk
Logical formatting or "making a file system"
Swap-space — Virtual memory uses disk space as an extension of main memory
RAID – redundant array of inexpensive disks multiple disk drives provides reliability via redundancy
RAID1: mirrored disks, RAID2: memory-style error-correcting codes (n+n-1). RAID3:bit-interleaved parity (n+1) RAID4: block-interleaved parity (n+1) RAID5: block-interleaved distributed parity.(n+1) RAID6:P+Q redundancy (n+2)

Device drivers encapsulate device details
Present uniform device-access interface to I/O subsystem
Controller (**host adapter**) – electronics that operate port, bus, device. Use busy bit in the register to show the status.
Polling: Read busy bit from status register until 0. Host sets read or write bit and if write copies data into data-out register. Host sets command-ready bit. Controller sets busy bit, executes transfer. Controller clears busy bit, error bit, command-ready bit when transfer done. inefficient if device slow
Interrupt-Driven I/O cycle: CPU device driver initiates I/O, I/O controller initiates I/O, input ready, output complete, or error generates interrupt signal. CPU receiving interrupt, transfers control to interrupt handler that processes data, returns from interrupt. CPU resumes processing of interrupted task
DMA: programmed I/O (one byte at a time) for large data movement, bypasses CPU to transfer data directly between I/O device and memory

1. Device driver is told to transfer disk data to buffer at address X
2. It tells dis controller to transfer C bytes from disk to buffer at X
3. Disk controller initiates DMA transfer
4. It sends each byte to DMA controller
5. DMA controller transfers bytes to X, increasing memory address and decreasing C until C = 0
6. DMA interrupts CPU to signal complete

Improving I/O Performance
Reduce number of context switches
Reduce data copying
Reduce interrupts by using large transfers, smart controllers, polling
Use DMA
Use smarter hardware devices
Balance CPU, memory, bus, and I/O performance for highest throughput
Move user-mode processes / daemons to kernel threads